



⑮

EUROPEAN PATENT APPLICATION

⑯ Application number: 89113847.1

⑯ Int. Cl. 4: G06F 3/033

⑯ Date of filing: 27.07.89

The title of the invention has been amended  
(Guidelines for Examination in the EPO, A-III,  
7.3).

⑯ Priority: 15.08.88 US 232353

⑯ Date of publication of application:  
28.02.90 Bulletin 90/09

⑯ Designated Contracting States:  
DE FR GB

⑯ Applicant: International Business Machines  
Corporation  
Old Orchard Road  
Armonk, N.Y. 10504(US)

⑯ Inventor: Gursky, Michael Carleton  
100 Cedar Grove Road  
Southbury Connecticut 06488(US)  
Inventor: Kelley, John Falk  
23 Powder Horn Road  
Peekskill New York 10566(US)

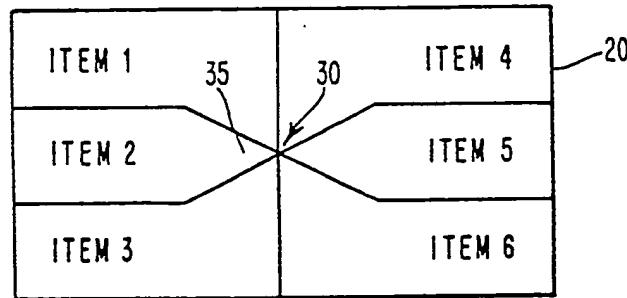
⑯ Representative: Blutke, Klaus, Dipl.-Ing.  
IBM Deutschland GmbH Intellectual Property  
Dept. Schönalcher Strasse 220  
D-7030 Böblingen(DE)

⑯ Method and apparatus for selecting items of a menu.

⑯ A method and apparatus for selecting any one (35) of a number of items of a menu (20) with minimal cursor movement. With this invention the items of the menu are displayed about a central point (30) with visible radii extending therefrom. When a user of the display terminal which is to make a selection for one of items of the menu, the user need merely move the cursor a minimal distance from the central point so that it is positioned in a corresponding area having two radii as sides of the area's perimeter. When the menu appears on the display, the cursor is positioned at the central point of the menu.

FIG. 1

EP 0 355 458 A2



## POP-UP MENU PIES

This invention relates to an apparatus and a method for selecting an item from a menu of a graphic display with minimal cursor movement, while also allowing for broader cursor movement by a novice terminal user. More specifically, the items of the display are displayed about a central point on the display with radii extending from that central point. An item is then selected by positioning the cursor anywhere in a corresponding area having two of the radii as sides of its perimeter.

In the past decade, the use of data processor controlled interactive display terminals has been widespread. In order to make the computer more friendly to the user, programmers have resorted to a use of menus which are displayed on the interactive display terminals. Traditional POP-UP menus on a mouse driven display system are typically presented as a vertical list choices. If the list is long, a fair amount of hand motion to select items on the menu is necessary.

A menu format which is not the traditional vertical list is disclosed in U.S. Patent 4,586,035 to Baker et al. This patent describes a multiple overlapping window type display on an interactive display terminal. One or more of the windows have an associated virtual distributed menu which is made of menu items which are distributed about the periphery of the associated window but not displayed during normal operations. When a cursor moves across a selected region in a window periphery, a selected menu item associated with the particular selected region will be displayed. Thus, it can be seen that the Baker patent requires that a cursor be moved about the periphery of a window in order to select an item in a menu. Moreover, the items on the menu are not visible.

U.S. Patent 4,692,858 to Redford et al discloses a method for assigning menu items to locations on a display. With this method, the first 16 items are placed in a single column, the next 16 items are split into two columns, the third set of 16 items are split into three columns, and so on. Selection of an item in a menu could require the user to move the cursor up to eight rows vertically and an indeterminate distance horizontally.

An article entitled, "An Empirical Comparison of Pie vs. Linear Menus" by Hopkins et al, discloses a menu in which the items are displayed in a circular pattern about the center of the circle. The previously mentioned article, however, does not use visible radii extending from a central point. Thus, using the menu described by Hopkins et al, the user of the menu is unaware of the effect of his next cursor movement. With the Hopkins article, some amount of guesswork as to where the cursor need be placed on the menu display in order to select a chosen item is required. With this invention, however, the user need not wander about the menu with the cursor in order to select a desired menu item. With this invention, the user knows exactly which item of the menu will be selected for any given cursor position. The aforementioned article was published in September 1987 from the Department of Computer Science of the University of Maryland.

Therefore, on a graphic display a need exists for a menu from which items can be selected with minimal cursor movement.

It is therefore an object of this invention to minimize cursor movement when selecting an item in a menu on a graphic display.

This object of the invention is accomplished by the features of the main claims. Further advantages of the invention are characterized in the subclaims.

The present invention provides a method and apparatus for selecting items of a menu on a graphic display with minimal cursor movement. This invention comprises means for displaying the items of a menu about a central point on a graphic display. This invention also comprises a means, responsive to the positioning of a cursor, for selecting one of the items when the cursor is positioned in an area between two radii extending from the central point.

The invention will be apparent from the following more particular description of preferred embodiments of the invention as illustrated in the accompanying figures.

FIG. 1 is a 6-entry menu with radii extending from a central point.

FIG. 2 is a 8-entry menu with radii extending from a central point.

FIG. 3 is a 10-entry menu with radii extending from a central point.

FIG. 4 is a 12-entry menu with radii extending from a central point.

FIG. 5 is a schematic illustration of an area with two parallel sides as part of its perimeter.

FIG. 6 is a schematic illustration of an area having two parallel sides orthogonal to a third side with all three sides being part of the perimeter of the area.

FIG. 7 is an arbitrary area with two radii extending from a central point.

FIG. 8 is a detailed construction of one half of an 8-item menu.

FIG. 9 is a detailed construction of one half of a 10-item menu.

FIG. 10 is a schematic illustration of an 11-item menu, six items in one column, five items in another column. Notice that the two columns are arranged about the central point.

FIG. 11 is a schematic illustration of an 11-item menu with six items in each column, with one of the items being unused in the right hand column.

5 FIGS. 12-15 illustrate various possible cases when testing for the intersection of a line with a line segment.

FIGS. 16 & 17 illustrate the values of a test condition relative to a line.

FIG. 18 is a schematic illustration of a circular menu with icons.

FIG. 19 is a flow chart for a procedure to generate the coordinates of a list of menu items.

10 FIG. 20 is a flow chart for a procedure to test whether a point is contained inside a polygon.

FIG. 21 is a flow chart for a procedure to handle the tracking of a pointing device and highlighting of menu items.

Shown in each of FIGS. 1-4 is a menu 20 of this invention which would appear on a graphic display. With the menu of FIG. 1, for example, a user of the display terminal can select any one of six items (see 15 items 1-6) on the menu. Shown also in each of FIGS. 1-4 is a central point 30 with radii 25 extending therefrom. Also shown is cursor 35 in a position between two radii corresponding to the selection of item 2 of the menu. Notice that with this menu, any item on the menu may be chosen with minimal or broad cursor movement. Typically, when the menu 20 pops up on a graphic display, the central point 30 appears at the 20 last cursor position, thereby minimizing reestablishment of the locus of attention. When, for example a user wishes to choose any one of the items of the menu, the user need merely move the cursor only a slight distance from the central point. For example, suppose the user decided to select item 8 of the menu. Since the cursor is typically at the central point, the user need merely move the cursor so that it appears in a corresponding area having two radii which form at least two sides of its perimeter. The area in this example corresponds to item 8.

25 Shown in FIGS. 5-7 are examples of areas that could be used to correspond to the selection of an item on a menu. Notice that all three areas have two radii 25 which extend from central point 30. Each pair of radii is also used to form the perimeter of a corresponding area. FIGS. 5 & 6 show an area with the perimeter also having two parallel sides 27 orthogonal to a third side 29. The use of the areas shown in 30 FIGS. 5 & 6 would facilitate the listing of items of a menu in a series of columns about a central point. FIG. 7 shows a more general type of area that could be used.

#### Hardware and System Software

35 This embodiment involves the use of an all-points-addressable (APA) display, which allows the drawing of diagonal lines, attached to a general-purpose personal computer. The display may be in color, may be monochrome, or may allow shades of gray.

The computer and attendant system software must have some method for displaying and moving an 40 on-screen cursor. The preferred pointing device for moving the cursor and also selecting menu items is a mouse or stylus, although the technique works with the use of keyboard keys alone. Application software must have the capability, through use of the underlying system software, to obtain input from the pointing device and to re-position the cursor at any desired location on the display.

The underlying system software is also assumed to enable the generation of at least a limited repertoire of graphic primitives, including lines, polygons and text strings.

45

#### Shape of Menu Entries

To be effective, this technique requires that two lines (radii) radiating from a central point in the menu 50 be included in the perimeter of the area corresponding to each item. The corresponding areas can otherwise be of arbitrary shape (see FIG. 7). For example, the entire menu could be circular (as illustrated in FIG. 18), with lines radiating from the center of the circle, making the corresponding areas resemble slices of a pie. This form might be appropriate when the menu items are to be labeled with icons.

The embodiment presented uses a rectangular menu, and the perimeter of each area is made up of 55 straight lines only. This simplifies the display of the menus and the recognition of which item has been selected. It also allows menu items to be labeled easily with text strings rather than icons. FIGS. 1, 2, 3, and 4 illustrate these menus with various numbers of items.

Menu Generation

The use of unit coordinates is assumed. That is, coordinate values are real numbers between 0 and 1, with (0,0) representing the lower left-hand corner of the screen and (1,1) the upper right-hand corner. These 5 are readily transformed back and forth from any real screen dimensions, making the algorithms presented here independent of the screen resolution.

The menus to be displayed can be generated once for a given application, or they might be dynamically created during the execution of a given application. The algorithm presented here is simple enough that it could be used in either circumstance.

10 Each menu will be stored as though centered at (0,0) and can be displayed at any location on the screen by adding a constant offset to the stored values.

In this preferred embodiment, a menu is represented as a list of menu items. The internal representation of each menu item contains at least the following information:

15 . the text of the label for the item  
. a list of the vertices of the polygonal perimeter of the area corresponding to the item.

The internal representation of a menu item might optionally contain other information such as:

20 . how to display the item (background color, background texture pattern, text color, etc.)  
. the status of the item (for example, whether it can currently be selected)  
. navigation (the name of another menu to display when this item is selected, or the name of a program or procedure to execute)

In addition, the internal representation of a menu will contain some global information:

25 . the coordinates of the upper-right corner of the menu

Procedure to generate menus: The following procedure "menu\_list" (see also FIG. 19) generates the coordinates of menu items. It returns an array of lists of points defining the polygonal perimeters of the 30 menu items. The items are ordered from top to bottom in the column, and each item's list of points is ordered in clockwise order.

```
100 procedure menu_list(h_item, w_label, n)
```

30

35

40

45

50

55

```

101  returns array of point_list

      begin
5       h_total = h_item * n

10      i_w = (n-1) / 2

15      y_w = h_total / 2 - i_w * h_item
           x_w = y_w * tan(i_w * π / n)

20      w_total = w + x_w

25      y_o = h_total / 2

30      107 and 107'    do i=1 to n

35      108      y_i = y_{i-1} - h_item
           x_i = y_i * tan(i * π / n)

40      109      menulist_i = (0.0, 0.0)
           ,
           110      if (x_{i-1}, y_{i-1}) ≠ (0.0, 0.0) then
           110'        append (x_{i-1}, y_{i-1}) to
           menulist_i

45      111      append (w_total, y_{i-1}) to menulist_i
           append (w_total, y_i) to menulist_i

50      112      if (x_i, y_i) ≠ (0.0, 0.0) then
           112'        append (x_i, y_i) to menulist_i

55      113      end do

56      114      return menulist
           end procedure

```

55 This algorithm produces coordinates as though the menu were centered at (0,0). A constant offset can be added and a coordinate transformation applied to obtain coordinates for displaying the menu at any desired location on the actual hardware screen.

The algorithm produces coordinates for the right-hand side of a menu only. The coordinates for the left-

hand side are obtained by negating the x coordinate values. If a menu is to contain an odd number of items, it would be possible to generate a menu containing an even number of items on one side and an odd number on the other. For example, a menu with 11 items might have six items on the left-hand side and five on the right-hand side (see FIG. 10). However, it may well be visually preferable to add an empty item 5 and to have the same number of items on each half of the menu. This would not necessarily require that an entry for the empty item actually be generated and maintained, just that the other items be generated as though the empty item existed. Thus a menu with 11 items might have six items on each half, with the last item on the right-hand side unused (see FIG. 11).

Description of the procedure: The function is given as parameters (code line 100):

- 10 . the height of each area corresponding to each item
- . the width of the labels (i.e. the length of the longest label to appear on the menu)
- . the number of items appearing on each side of the menu.

The height and width are positive real numbers, and the number of items is a positive integer.

The procedure returns an array, each element of which is a list of (x,y) coordinate pairs (code lines 101, 15 114).

The total height of the menu is the height of each area corresponding to each item times the number of items (code line 102). The total width is the width of the labels plus the rightmost endpoint ( $x_w$ ) of any of the lines radiating from the central point (code line 105). See FIG. 8 for an illustration of the points involved.  $i_w$  is the index of the item containing this rightmost endpoint (code line 103). Note that this is an integer, so 20 that if  $n$  is even, the value  $(n - 1)/2$  is truncated. The point  $(x_w, y_w)$  is the intersection of the radiating line  $l_w$  with the horizontal line at  $y_w$  (code line 104).

Initial values for the loop (code lines 107-113) are set up in code line 106. The loop is repeated for each menu item (code line 107). Code line 108 causes items to be generated from top (higher y values) to bottom.  $x_i$  is again the intersection of the radiating line with the horizontal line at  $y_i$ .

25 menulist; is the list of coordinate points for menu item number  $i$ . All lists begin with the central point (0,0) (code line 109). See FIGS. 8 and 9 for illustrations of menus with both even and odd numbers of items. Note that some items contain four sides and some contain five. The tests at code lines 110 and 112 provide that duplicate points are not included in the list. The points at the right end of each item are included in the list in code line 111. There is an implied side from the last point on the list back to (0,0) (the 30 first point on the list).

#### Menu Placement and Display

35 The preferred location for a menu which will be displayed and subsequently removed, typically after a choice has been made from the menu, is as close as possible to the current on-screen cursor location at the moment the display of the menu is initiated. This allows the user's attention to remain focused at the site of the cursor and minimizes the motion required by the hand controlling the pointing device.

If the current cursor location is sufficiently far from the margins of the screen, the menu can be 40 centered exactly at the cursor location. Otherwise, the center of the menu will have to be moved to allow the entire menu to be shown on-screen.

In particular if the current cursor location is  $(x_c, y_c)$  and the upper-right corner of the menu as stored is at  $(x_m, y_m)$ , then the offset which should be added to the stored menu coordinates in order to display the menu is:

45  $(\min(\max(x_c, x_m), 1-x_c), \min(\max(y_c, y_m), 1-y_c))$

After the menu has been displayed, the cursor is moved to (redisplays at) the center of the menu. This is essential to allow all choices to be made with an equal motion of the pointing device. After the menu is removed from the screen, the cursor may be moved back to the location it was at before the menu was 50 displayed, or it may remain at the location it was at immediately before the menu was removed.

As an alternative to displaying menus at or near the current cursor location, the menus may be displayed always at the same location on the screen, for example one of the corners. In this case, the only requirement to gain the benefit of the menus described here is that the cursor be moved to the center of the menu after it is displayed. This may, however, require a shift of attention by the user from one area of 55 the screen to another.

The actual display of the menu may be done in a variety of ways, so long as the polygonal perimeter of each menu item, especially the two radii extending from the central point of the menu, is visible to the user. For example, the perimeter of each item might be drawn with lines, or the body of each item might be

drawn as a solid or patterned polygon. In the latter case, the radii extending from the central point are visible as boundaries between the areas enclosing menu items.

## 5 Detection of Inclusion in Menu Items

It is desirable to highlight the current menu item (i.e. the menu item, if any, whose corresponding area contains the current cursor location) as the user moves the cursor on the screen, so that the user has a constant visual reminder of which item would be chosen were a selection to be made at that cursor location.

10 Once a selection has been made (for example, a mouse button pressed), it may be necessary to determine in which menu item the cursor location lies. Since the menu items generated by "menu\_list" contain only straight sides, it suffices to have a test for inclusion of a point in a polygon.

15 Procedure to test for inclusion in a polygon: The following procedure "test\_inside" tests whether a point is contained inside a polygon (see also FIG. 20). The algorithm is based upon the well-known mathematical fact that a straight line extended from a point in a plane will intersect a closed figure in the same plane an odd number of times if the point lies inside the curve and an even number of times if it lies outside the curve. This procedure uses a line extending horizontally to the left from the point in question.

```
200  procedure test_inside(poly, p)
201      returns integer
202
203      begin
204          n = number of points in poly
205
206          for i = 1 to n
207              if point_in_poly(poly, p, i) then
208                  return 1
209
210          return 0
211
212      end
213
214      point_in_poly(poly, p, i)
215      returns boolean
216
217      begin
218          for j = 1 to n
219              if j = i then
220                  continue
221
222              if line_crosses_line(poly, p, i, j) then
223                  if odd(j) then
224                      return true
225                  else
226                      return false
227
228          return false
229
230      end
231
232      line_crosses_line(poly, p, i, j)
233      returns boolean
234
235      begin
236          x1 = point_x(poly, i)
237          y1 = point_y(poly, i)
238          x2 = point_x(poly, j)
239          y2 = point_y(poly, j)
240
241          x3 = point_x(p, i)
242          y3 = point_y(p, i)
243          x4 = point_x(p, j)
244          y4 = point_y(p, j)
245
246          if (x1 - x3) * (y4 - y3) - (x1 - x4) * (y3 - y1) = 0 then
247              return false
248
249          if (x1 - x3) * (y4 - y3) < 0 then
250              if (x1 - x3) * (y2 - y1) < 0 then
251                  if (x1 - x3) * (y1 - y3) < 0 then
252                      return true
253                  else
254                      return false
255
256          if (x1 - x3) * (y4 - y3) > 0 then
257              if (x1 - x3) * (y2 - y1) > 0 then
258                  if (x1 - x3) * (y1 - y3) > 0 then
259                      return false
260                  else
261                      return true
262
263          return false
264
265      end
266
267      point_x(poly, i)
268      returns real
269
270      begin
271          return point_x(poly, i)
272
273      end
274
275      point_y(poly, i)
276      returns real
277
278      begin
279          return point_y(poly, i)
280
281      end
282
283      point_x(p, i)
284      returns real
285
286      begin
287          return point_x(p, i)
288
289      end
290
291      point_y(p, i)
292      returns real
293
294      begin
295          return point_y(p, i)
296
297      end
298
299      odd(i)
300      returns boolean
301
302      begin
303          if i % 2 = 1 then
304              return true
305          else
306              return false
307
308      end
309
310      even(i)
311      returns boolean
312
313      begin
314          if i % 2 = 0 then
315              return true
316          else
317              return false
318
319      end
320
321      point_in_poly(poly, p, i)
322      returns boolean
323
324      begin
325          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
326              return true
327
328          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
329              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
330                  return true
331
332          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
333              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
334                  return true
335
336          return false
337
338      end
339
340      point_x(poly, i)
341      returns real
342
343      begin
344          return point_x(poly, i)
345
346      end
347
348      point_y(poly, i)
349      returns real
350
351      begin
352          return point_y(poly, i)
353
354      end
355
356      point_x(p, i)
357      returns real
358
359      begin
360          return point_x(p, i)
361
362      end
363
364      point_y(p, i)
365      returns real
366
367      begin
368          return point_y(p, i)
369
370      end
371
372      line_crosses_line(poly, p, i, j)
373      returns boolean
374
375      begin
376          if (x1 - x3) * (y4 - y3) - (x1 - x4) * (y3 - y1) = 0 then
377              return false
378
379          if (x1 - x3) * (y4 - y3) < 0 then
380              if (x1 - x3) * (y2 - y1) < 0 then
381                  if (x1 - x3) * (y1 - y3) < 0 then
382                      return true
383                  else
384                      return false
385
386          if (x1 - x3) * (y4 - y3) > 0 then
387              if (x1 - x3) * (y2 - y1) > 0 then
388                  if (x1 - x3) * (y1 - y3) > 0 then
389                      return false
390                  else
391                      return true
392
393          return false
394
395      end
396
397      point_x(poly, i)
398      returns real
399
400      begin
401          return point_x(poly, i)
402
403      end
404
405      point_y(poly, i)
406      returns real
407
408      begin
409          return point_y(poly, i)
410
411      end
412
413      point_x(p, i)
414      returns real
415
416      begin
417          return point_x(p, i)
418
419      end
420
421      point_y(p, i)
422      returns real
423
424      begin
425          return point_y(p, i)
426
427      end
428
429      odd(i)
430      returns boolean
431
432      begin
433          if i % 2 = 1 then
434              return true
435          else
436              return false
437
438      end
439
440      even(i)
441      returns boolean
442
443      begin
444          if i % 2 = 0 then
445              return true
446          else
447              return false
448
449      end
450
451      point_in_poly(poly, p, i)
452      returns boolean
453
454      begin
455          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
456              return true
457
458          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
459              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
460                  return true
461
462          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
463              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
464                  return true
465
466          return false
467
468      end
469
470      point_x(poly, i)
471      returns real
472
473      begin
474          return point_x(poly, i)
475
476      end
477
478      point_y(poly, i)
479      returns real
480
481      begin
482          return point_y(poly, i)
483
484      end
485
486      point_x(p, i)
487      returns real
488
489      begin
490          return point_x(p, i)
491
492      end
493
494      point_y(p, i)
495      returns real
496
497      begin
498          return point_y(p, i)
499
500      end
501
502      line_crosses_line(poly, p, i, j)
503      returns boolean
504
505      begin
506          if (x1 - x3) * (y4 - y3) - (x1 - x4) * (y3 - y1) = 0 then
507              return false
508
509          if (x1 - x3) * (y4 - y3) < 0 then
510              if (x1 - x3) * (y2 - y1) < 0 then
511                  if (x1 - x3) * (y1 - y3) < 0 then
512                      return true
513                  else
514                      return false
515
516          if (x1 - x3) * (y4 - y3) > 0 then
517              if (x1 - x3) * (y2 - y1) > 0 then
518                  if (x1 - x3) * (y1 - y3) > 0 then
519                      return false
520                  else
521                      return true
522
523          return false
524
525      end
526
527      point_x(poly, i)
528      returns real
529
530      begin
531          return point_x(poly, i)
532
533      end
534
535      point_y(poly, i)
536      returns real
537
538      begin
539          return point_y(poly, i)
540
541      end
542
543      point_x(p, i)
544      returns real
545
546      begin
547          return point_x(p, i)
548
549      end
550
551      point_y(p, i)
552      returns real
553
554      begin
555          return point_y(p, i)
556
557      end
558
559      odd(i)
560      returns boolean
561
562      begin
563          if i % 2 = 1 then
564              return true
565          else
566              return false
567
568      end
569
570      even(i)
571      returns boolean
572
573      begin
574          if i % 2 = 0 then
575              return true
576          else
577              return false
578
579      end
580
581      point_in_poly(poly, p, i)
582      returns boolean
583
584      begin
585          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
586              return true
587
588          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
589              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
590                  return true
591
592          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
593              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
594                  return true
595
596          return false
597
598      end
599
600      point_x(poly, i)
601      returns real
602
603      begin
604          return point_x(poly, i)
605
606      end
607
608      point_y(poly, i)
609      returns real
610
611      begin
612          return point_y(poly, i)
613
614      end
615
616      point_x(p, i)
617      returns real
618
619      begin
620          return point_x(p, i)
621
622      end
623
624      point_y(p, i)
625      returns real
626
627      begin
628          return point_y(p, i)
629
630      end
631
632      odd(i)
633      returns boolean
634
635      begin
636          if i % 2 = 1 then
637              return true
638          else
639              return false
640
641      end
642
643      even(i)
644      returns boolean
645
646      begin
647          if i % 2 = 0 then
648              return true
649          else
650              return false
651
652      end
653
654      point_in_poly(poly, p, i)
655      returns boolean
656
657      begin
658          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
659              return true
660
661          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
662              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
663                  return true
664
665          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
666              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
667                  return true
668
669          return false
670
671      end
672
673      point_x(poly, i)
674      returns real
675
676      begin
677          return point_x(poly, i)
678
679      end
680
681      point_y(poly, i)
682      returns real
683
684      begin
685          return point_y(poly, i)
686
687      end
688
689      point_x(p, i)
690      returns real
691
692      begin
693          return point_x(p, i)
694
695      end
696
697      point_y(p, i)
698      returns real
699
700      begin
701          return point_y(p, i)
702
703      end
704
705      odd(i)
706      returns boolean
707
708      begin
709          if i % 2 = 1 then
710              return true
711          else
712              return false
713
714      end
715
716      even(i)
717      returns boolean
718
719      begin
720          if i % 2 = 0 then
721              return true
722          else
723              return false
724
725      end
726
727      point_in_poly(poly, p, i)
728      returns boolean
729
730      begin
731          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
732              return true
733
734          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
735              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
736                  return true
737
738          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
739              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
740                  return true
741
742          return false
743
744      end
745
746      point_x(poly, i)
747      returns real
748
749      begin
750          return point_x(poly, i)
751
752      end
753
754      point_y(poly, i)
755      returns real
756
757      begin
758          return point_y(poly, i)
759
760      end
761
762      point_x(p, i)
763      returns real
764
765      begin
766          return point_x(p, i)
767
768      end
769
770      point_y(p, i)
771      returns real
772
773      begin
774          return point_y(p, i)
775
776      end
777
778      odd(i)
779      returns boolean
780
781      begin
782          if i % 2 = 1 then
783              return true
784          else
785              return false
786
787      end
788
789      even(i)
790      returns boolean
791
792      begin
793          if i % 2 = 0 then
794              return true
795          else
796              return false
797
798      end
799
800      point_in_poly(poly, p, i)
801      returns boolean
802
803      begin
804          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
805              return true
806
807          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
808              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
809                  return true
810
811          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
812              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
813                  return true
814
815          return false
816
817      end
818
819      point_x(poly, i)
820      returns real
821
822      begin
823          return point_x(poly, i)
824
825      end
826
827      point_y(poly, i)
828      returns real
829
830      begin
831          return point_y(poly, i)
832
833      end
834
835      point_x(p, i)
836      returns real
837
838      begin
839          return point_x(p, i)
840
841      end
842
843      point_y(p, i)
844      returns real
845
846      begin
847          return point_y(p, i)
848
849      end
850
851      odd(i)
852      returns boolean
853
854      begin
855          if i % 2 = 1 then
856              return true
857          else
858              return false
859
860      end
861
862      even(i)
863      returns boolean
864
865      begin
866          if i % 2 = 0 then
867              return true
868          else
869              return false
870
871      end
872
873      point_in_poly(poly, p, i)
874      returns boolean
875
876      begin
877          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
878              return true
879
880          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
881              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
882                  return true
883
884          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
885              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
886                  return true
887
888          return false
889
890      end
891
892      point_x(poly, i)
893      returns real
894
895      begin
896          return point_x(poly, i)
897
898      end
899
900      point_y(poly, i)
901      returns real
902
903      begin
904          return point_y(poly, i)
905
906      end
907
908      point_x(p, i)
909      returns real
910
911      begin
912          return point_x(p, i)
913
914      end
915
916      point_y(p, i)
917      returns real
918
919      begin
920          return point_y(p, i)
921
922      end
923
924      odd(i)
925      returns boolean
926
927      begin
928          if i % 2 = 1 then
929              return true
930          else
931              return false
932
933      end
934
935      even(i)
936      returns boolean
937
938      begin
939          if i % 2 = 0 then
940              return true
941          else
942              return false
943
944      end
945
946      point_in_poly(poly, p, i)
947      returns boolean
948
949      begin
950          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
951              return true
952
953          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
954              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
955                  return true
956
957          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
958              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
959                  return true
960
961          return false
962
963      end
964
965      point_x(poly, i)
966      returns real
967
968      begin
969          return point_x(poly, i)
970
971      end
972
973      point_y(poly, i)
974      returns real
975
976      begin
977          return point_y(poly, i)
978
979      end
980
981      point_x(p, i)
982      returns real
983
984      begin
985          return point_x(p, i)
986
987      end
988
989      point_y(p, i)
990      returns real
991
992      begin
993          return point_y(p, i)
994
995      end
996
997      odd(i)
998      returns boolean
999
1000      begin
1001          if i % 2 = 1 then
1002              return true
1003          else
1004              return false
1005
1006      end
1007
1008      even(i)
1009      returns boolean
1010
1011      begin
1012          if i % 2 = 0 then
1013              return true
1014          else
1015              return false
1016
1017      end
1018
1019      point_in_poly(poly, p, i)
1020      returns boolean
1021
1022      begin
1023          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1024              return true
1025
1026          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1027              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1028                  return true
1029
1030          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1031              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1032                  return true
1033
1034          return false
1035
1036      end
1037
1038      point_x(poly, i)
1039      returns real
1040
1041      begin
1042          return point_x(poly, i)
1043
1044      end
1045
1046      point_y(poly, i)
1047      returns real
1048
1049      begin
1050          return point_y(poly, i)
1051
1052      end
1053
1054      point_x(p, i)
1055      returns real
1056
1057      begin
1058          return point_x(p, i)
1059
1060      end
1061
1062      point_y(p, i)
1063      returns real
1064
1065      begin
1066          return point_y(p, i)
1067
1068      end
1069
1070      odd(i)
1071      returns boolean
1072
1073      begin
1074          if i % 2 = 1 then
1075              return true
1076          else
1077              return false
1078
1079      end
1080
1081      even(i)
1082      returns boolean
1083
1084      begin
1085          if i % 2 = 0 then
1086              return true
1087          else
1088              return false
1089
1090      end
1091
1092      point_in_poly(poly, p, i)
1093      returns boolean
1094
1095      begin
1096          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1097              return true
1098
1099          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1100              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1101                  return true
1102
1103          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1104              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1105                  return true
1106
1107          return false
1108
1109      end
1110
1111      point_x(poly, i)
1112      returns real
1113
1114      begin
1115          return point_x(poly, i)
1116
1117      end
1118
1119      point_y(poly, i)
1120      returns real
1121
1122      begin
1123          return point_y(poly, i)
1124
1125      end
1126
1127      point_x(p, i)
1128      returns real
1129
1130      begin
1131          return point_x(p, i)
1132
1133      end
1134
1135      point_y(p, i)
1136      returns real
1137
1138      begin
1139          return point_y(p, i)
1140
1141      end
1142
1143      odd(i)
1144      returns boolean
1145
1146      begin
1147          if i % 2 = 1 then
1148              return true
1149          else
1150              return false
1151
1152      end
1153
1154      even(i)
1155      returns boolean
1156
1157      begin
1158          if i % 2 = 0 then
1159              return true
1160          else
1161              return false
1162
1163      end
1164
1165      point_in_poly(poly, p, i)
1166      returns boolean
1167
1168      begin
1169          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1170              return true
1171
1172          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1173              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1174                  return true
1175
1176          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1177              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1178                  return true
1179
1180          return false
1181
1182      end
1183
1184      point_x(poly, i)
1185      returns real
1186
1187      begin
1188          return point_x(poly, i)
1189
1190      end
1191
1192      point_y(poly, i)
1193      returns real
1194
1195      begin
1196          return point_y(poly, i)
1197
1198      end
1199
1200      point_x(p, i)
1201      returns real
1202
1203      begin
1204          return point_x(p, i)
1205
1206      end
1207
1208      point_y(p, i)
1209      returns real
1210
1211      begin
1212          return point_y(p, i)
1213
1214      end
1215
1216      odd(i)
1217      returns boolean
1218
1219      begin
1220          if i % 2 = 1 then
1221              return true
1222          else
1223              return false
1224
1225      end
1226
1227      even(i)
1228      returns boolean
1229
1230      begin
1231          if i % 2 = 0 then
1232              return true
1233          else
1234              return false
1235
1236      end
1237
1238      point_in_poly(poly, p, i)
1239      returns boolean
1240
1241      begin
1242          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1243              return true
1244
1245          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1246              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1247                  return true
1248
1249          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1250              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1251                  return true
1252
1253          return false
1254
1255      end
1256
1257      point_x(poly, i)
1258      returns real
1259
1260      begin
1261          return point_x(poly, i)
1262
1263      end
1264
1265      point_y(poly, i)
1266      returns real
1267
1268      begin
1269          return point_y(poly, i)
1270
1271      end
1272
1273      point_x(p, i)
1274      returns real
1275
1276      begin
1277          return point_x(p, i)
1278
1279      end
1280
1281      point_y(p, i)
1282      returns real
1283
1284      begin
1285          return point_y(p, i)
1286
1287      end
1288
1289      odd(i)
1290      returns boolean
1291
1292      begin
1293          if i % 2 = 1 then
1294              return true
1295          else
1296              return false
1297
1298      end
1299
1300      even(i)
1301      returns boolean
1302
1303      begin
1304          if i % 2 = 0 then
1305              return true
1306          else
1307              return false
1308
1309      end
1310
1311      point_in_poly(poly, p, i)
1312      returns boolean
1313
1314      begin
1315          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1316              return true
1317
1318          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1319              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1320                  return true
1321
1322          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1323              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1324                  return true
1325
1326          return false
1327
1328      end
1329
1330      point_x(poly, i)
1331      returns real
1332
1333      begin
1334          return point_x(poly, i)
1335
1336      end
1337
1338      point_y(poly, i)
1339      returns real
1340
1341      begin
1342          return point_y(poly, i)
1343
1344      end
1345
1346      point_x(p, i)
1347      returns real
1348
1349      begin
1350          return point_x(p, i)
1351
1352      end
1353
1354      point_y(p, i)
1355      returns real
1356
1357      begin
1358          return point_y(p, i)
1359
1360      end
1361
1362      odd(i)
1363      returns boolean
1364
1365      begin
1366          if i % 2 = 1 then
1367              return true
1368          else
1369              return false
1370
1371      end
1372
1373      even(i)
1374      returns boolean
1375
1376      begin
1377          if i % 2 = 0 then
1378              return true
1379          else
1380              return false
1381
1382      end
1383
1384      point_in_poly(poly, p, i)
1385      returns boolean
1386
1387      begin
1388          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1389              return true
1390
1391          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1392              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1393                  return true
1394
1395          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1396              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1397                  return true
1398
1399          return false
1400
1401      end
1402
1403      point_x(poly, i)
1404      returns real
1405
1406      begin
1407          return point_x(poly, i)
1408
1409      end
1410
1411      point_y(poly, i)
1412      returns real
1413
1414      begin
1415          return point_y(poly, i)
1416
1417      end
1418
1419      point_x(p, i)
1420      returns real
1421
1422      begin
1423          return point_x(p, i)
1424
1425      end
1426
1427      point_y(p, i)
1428      returns real
1429
1430      begin
1431          return point_y(p, i)
1432
1433      end
1434
1435      odd(i)
1436      returns boolean
1437
1438      begin
1439          if i % 2 = 1 then
1440              return true
1441          else
1442              return false
1443
1444      end
1445
1446      even(i)
1447      returns boolean
1448
1449      begin
1450          if i % 2 = 0 then
1451              return true
1452          else
1453              return false
1454
1455      end
1456
1457      point_in_poly(poly, p, i)
1458      returns boolean
1459
1460      begin
1461          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1462              return true
1463
1464          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1465              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1466                  return true
1467
1468          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1469              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1470                  return true
1471
1472          return false
1473
1474      end
1475
1476      point_x(poly, i)
1477      returns real
1478
1479      begin
1480          return point_x(poly, i)
1481
1482      end
1483
1484      point_y(poly, i)
1485      returns real
1486
1487      begin
1488          return point_y(poly, i)
1489
1490      end
1491
1492      point_x(p, i)
1493      returns real
1494
1495      begin
1496          return point_x(p, i)
1497
1498      end
1499
1500      point_y(p, i)
1501      returns real
1502
1503      begin
1504          return point_y(p, i)
1505
1506      end
1507
1508      odd(i)
1509      returns boolean
1510
1511      begin
1512          if i % 2 = 1 then
1513              return true
1514          else
1515              return false
1516
1517      end
1518
1519      even(i)
1520      returns boolean
1521
1522      begin
1523          if i % 2 = 0 then
1524              return true
1525          else
1526              return false
1527
1528      end
1529
1530      point_in_poly(poly, p, i)
1531      returns boolean
1532
1533      begin
1534          if point_x(poly, i) = point_x(p, i) and point_y(poly, i) = point_y(p, i) then
1535              return true
1536
1537          if point_x(poly, i) < point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1538              if point_x(poly, i) < point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1539                  return true
1540
1541          if point_x(poly, i) > point_x(p, i) and point_y(poly, i) < point_y(p, i) then
1542              if point_x(poly, i) > point_x(p, i) and point_y(poly, i) > point_y(p, i) then
1543                  return true
1544
1545          return false
1546
1547      end
1548
1549      point_x(poly, i)
1550      returns real
1551
1552      begin
1553          return point_x(poly, i)
1554
1555      end
1556
1557      point_y(poly, i)
1558      returns real
1559
1560      begin
1561          return point_y(poly, i)
1562
1563      end
1564
1565      point_x(p, i)
1566      returns
```

```

203  p2 = polyn
5
204  inside = 0

205 and 205'  do i=1 to n

10  206      p1 = p2
          p2 = polyi

15  207      if p1.y ≥ p.y and p2.y ≥ p.y then
          next i

20  208      if p1.y ≤ p.y and p2.y ≤ p.y then
          next i

25  209      if p1.x ≥ p.x and p2.x ≥ p.x then
          next i

30  210      if p1.x < p.x and p2.x < p.x then
          begin
35.  211          inside = 1 - inside
          next i
          end if

40  212      test = (p.x - p1.x)*(p2.y - p1.y) -
          (p.y - p1.y)*(p2.x - p1.x)

45  213      if (p2.y > p1.y and test > 0) or
          (p2.y < p1.y and test < 0) then
45.  213'          inside = 1 - inside

50  214      end do

50  215      return inside
      end procedure

```

56 Description of the procedure: The procedure is passed as parameters (code line 200):  
 . a polygon "poly", represented as a list of points with the last point connecting to the first point to close the polygon  
 . a point "p".

The procedure returns an integer value which will be 0 if the point lies outside the polygon and 1 if it lies inside the polygon (code line 201).

The number of points in the polygon list is either stored explicitly as part of the data structure representing the list or can be obtained by scanning through the list, counting the points (code line 202).

5 The algorithm considers the intersection of the line extending from p with successive line segments extending from point p1 to point p2. The endpoint p2 is initialized to the last point in the polygon point list (code line 203) so that the first line segment considered will be from the last point to the first.

The variable "inside" will contain the parity of the number of crossings seen so far (0 if even, 1 if odd). Initially, no line crossings have been detected (code line 204).

10 The loop (code lines 205-214) is repeated for each line segment in the polygon. The previous endpoint becomes the current startpoint, and the current endpoint is the next point in the polygon list (code line 206).

The notations ".x" and ".y" signify the x and y components of the point in question. The statement "next i" means that the next iteration of the loop will be started immediately without performing the remaining steps in the body of the loop.

15 See FIGS. 12-17 for an illustration of the various cases being tested for in code lines 207-213. If p lies below the line segment, there can be no crossing (code line 207; see FIG. 12). If it lies above the segment, there can be no crossing (code line 208; see FIG. 13). If it lies to the left of the segment, there can be no crossing (code line 209; see FIG. 14). If it lies to the right of the line segment (and is known not to be either above or below), then there must be a crossing (code line 210 and 211; see FIG. 15). The last two tests are

20 not strictly necessary, since the test value calculated in code line 212 is general. However, performing them reduces the number of times the more expensive calculation in code line 212 needs to be done.

Finally, the relationship between the point p and the line from p1 to p may have to be calculated. The test value (code line 212) is derived from the formula for the line passing between point p1 and p2. If the line segment (directed from p1 to p2) extends upward and the test value is positive (see FIG. 16), or if the line segment extends downward and the test value is negative (see FIG. 17), the point lies to the right of the line and a crossing occurs (code line 213 and 213').

The parity after all segments have been tested will reflect whether the point lies inside or outside the polygon (code line 215).

30

#### Tracking and Recognition of a Selection

As mentioned above, it is desirable to interactively highlight menu items as the cursor tracks the user's movement of the pointing device.

35 Procedure to highlight menu items: The following procedure "track" handles the tracking of the pointing device and highlighting of the menu items see also FIG. 21). It assumes that only one menu is currently displayed on the screen, but can be generalized in a straightforward manner to handle multiple menus.

```

40      300  procedure track(menuList)
41
42          301  returns integer
43
44          45      begin
45          302      n = number of items in menuList
46
47          50      303  current = 0
48
49          304  do forever
50
51          55      305      p = new cursor location
52              display cursor at location p

```

```

306      newcurrent = 0

5       307 and 307'      do i=1 to n

10      308      if test_inside(menuListi, p) ≠ 0
           then
           begin
15      309      newcurrent = i
           exitdo
           end if
           310      end do

20      311      if current ≠ newcurrent then
           begin
           312      if current ≠ 0 then
           312'      unhighlight(menuListcurrent)
25      313      current = newcurrent

30      314      if current ≠ 0 then
           314'      highlight(menuListcurrent)
           end if

35      315      if current ≠ 0 and selection made then
           exitdo

40      316      end do

45      317      return current
           end procedure

```

Description of the procedure: The procedure is passed as a parameter (code line 300): a list of menu items, as generated with the aid of procedure "menu\_list".

50 The procedure returns the index of the item selected (code line 301). As written, the procedure does not terminate until some valid selection is made. The test at code line 315 can easily be expanded to test for any exceptional inputs or conditions which might cause the procedure to terminate without a valid choice, in which case the return value would be 0.

55 The number of items in the menu list is either stored explicitly as part of the data structure representing the list or can be obtained by scanning through the list, counting the items (code line 302). Initially, no menu item is selected or highlighted (code line 303).

The outer loop (code line 304-316) is executed indefinitely until explicitly terminated. The notation "exitdo" signifies that the nearest enclosing do loop is terminated. Any additional outer loops are not

terminated.

The new cursor location is obtained, and the on-screen cursor moved to that location (code line 305). The underlying system software is assumed to employ well-known techniques for handling interrupts from the pointing device hardware, applying hysteresis to the input if necessary, translating to a desired coordinate system if necessary, and storing the values in an input buffer, allowing application programs to retrieve the succession of values from the input buffer.

5 The new cursor location is initially assumed to lie outside all menu items (code line 306). The loop at code lines 307-310 checks whether it lies inside the perimeter of any item. The procedure "test\_inside" is called for each menu item to determine if the point lies inside the perimeter of the item (code line 308). If 10 so, the index is remembered and the search terminates (code line 309).

10 Once the new current item, if any, has been determined, highlights may have to be removed or added to the previous and new current items (code lines 311-314). This needs to be done only if the current items have changed (code line 311).

15 Highlighting of a menu item can be done in a number of ways, for example by changing the colors of the items or by drawing a bounding polygon around the item. The specifics of highlighting and removing the highlight from an item are not given here.

15 Once any highlight has been removed (code line 312 and 312'), the old current item can be forgotten (code line 313) and the new one highlighted if it exists (code line 314 and 314').

20 If a selection has been made (for example, a mouse button pressed) and the cursor is in fact inside the perimeter of some menu item, then tracking can terminate (code line 315). If the cursor is not inside the perimeter of any menu item, tracking continues.

Once the indeterminate loop is exited, "current" contains the index of the selected menu item (code line 317).

25 **Claims**

1. A graphic display wherein any one of a number of items of a menu is selected by positioning a cursor on said display, an apparatus for selecting one of said items with minimal or broad cursor movement 30 comprising:

means for displaying said items about only one central point on said display, with said cursor being simultaneously positioned at said central point when said items are displayed about said central point; and means, responsive to positioning of said cursor, for selecting one of said items when said cursor is positioned in a corresponding area having two radii as sides of its perimeter with said radii extending from 35 said central point and with said radii being displayed on said display.

2. An apparatus as claimed in claim 1, wherein said displaying means places said central point where said cursor last appeared on said display.

3. An apparatus as claimed in claim 1, wherein said displaying means displays said items in two columns about a central point, and wherein each said area has a corresponding one of said items displayed 40 therein.

4. An apparatus as claimed in claim 1, wherein each said area has two parallel sides as part of the perimeter thereof, said sides being orthogonal to a third side of said perimeter.

5. A graphic display wherein any one of a number of items of a menu is selected by positioning a cursor on said display, an apparatus for selecting one of said items with minimal or broad cursor movement 45 comprising:

means for displaying said items in two columns about only one central point on said display, with said cursor being simultaneously positioned at said central point when said items are displayed about said central point; and means, responsive to positioning of said cursor, for selecting one of said items when said cursor is positioned in a corresponding area having two radii as sides of its perimeter with said radii extending from 50 said central point, with said radii being displayed on said display, and with each of said items being displayed in a corresponding said area.

6. An apparatus as claimed in claim 5, wherein said displaying means places said central point where said cursor last appeared on said display.

55 7. An apparatus as claimed in claim 6, wherein each said area has two parallel sides as part of the perimeter thereof, said sides being orthogonal to a third side of said perimeter.

8. A selection method for a graphic display, wherein any one of a number of items of a menu is selected by positioning a cursor on said display, selection one of said items being made with minimal or

broad cursor movement, comprising the steps of:  
displaying on said display said items about only one central point on said display, with said cursor being simultaneously positioned at said central point when said items are displayed about said central point; and  
positioning said cursor, for selecting one of said items when said cursor is positioned in a corresponding area having two radii as sides of its perimeter with said radii extending from said central point and with said radii being displayed on said display.

9. A method as claimed in claim 8, wherein said central point is displayed at the position where said cursor last appeared on said display.

10. A method as claimed in claim 8, wherein said items are displayed in two columns about a central point with each said area having a corresponding one of said items displayed therein.

11. A method as claimed in claim 8, wherein each said area has two parallel sides as part of the perimeter thereof with said sides being orthogonal to a third side of said perimeter.

12. A selection method for a graphic display, wherein any one of a number of items of a menu is selected by positioning a cursor on said display, selection one of said items being made with minimal or broad cursor movement, comprising the steps of:  
displaying on said display said items in two columns about only one central point on said display, with said cursor being simultaneously positioned at said central point when said items are displayed about said central point; and  
positioning said cursor, for selecting one of said items when said cursor is positioned in a corresponding area having two radii as sides of its perimeter with said radii extending from said central point and with said radii being displayed on said display, and with each of said items being displayed in a corresponding said area.

13. A method as claimed in claim 12, wherein said central point is displayed at the position where said cursor last appeared on said display.

25

30

35

40

45

50

55

FIG. 1

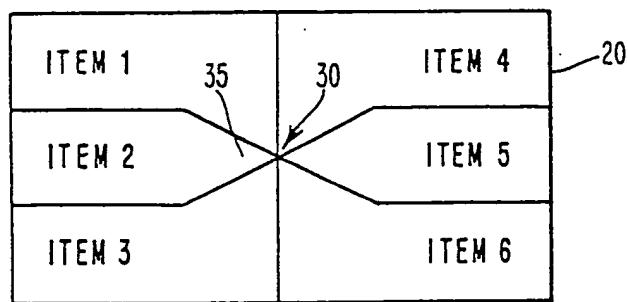


FIG. 2

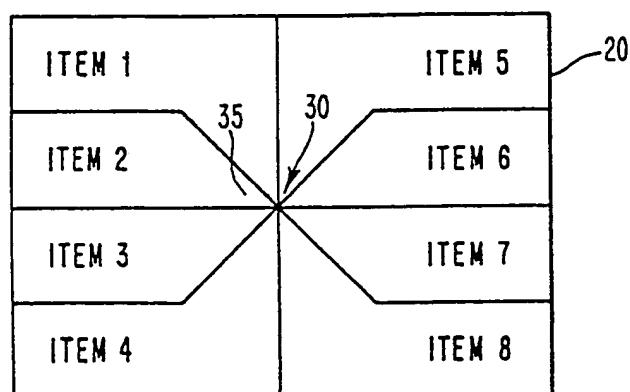


FIG. 3

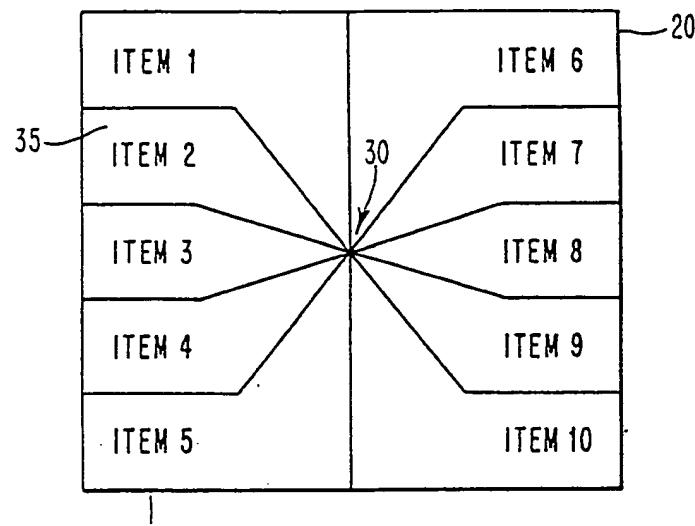


FIG. 4

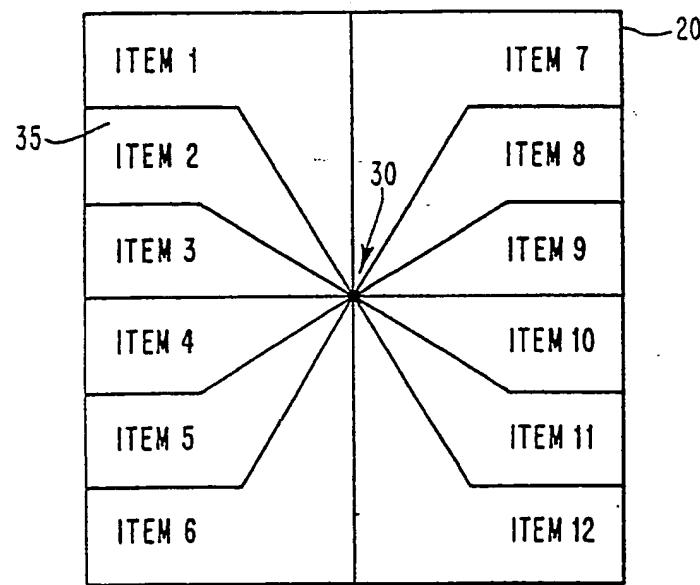


FIG. 5

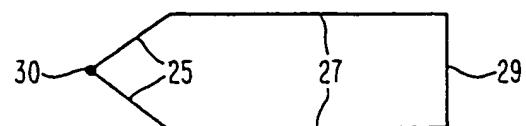


FIG. 6

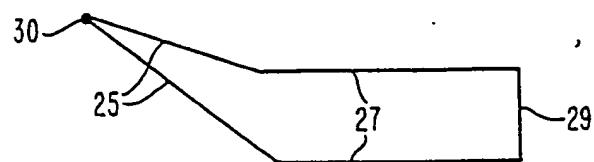


FIG. 7

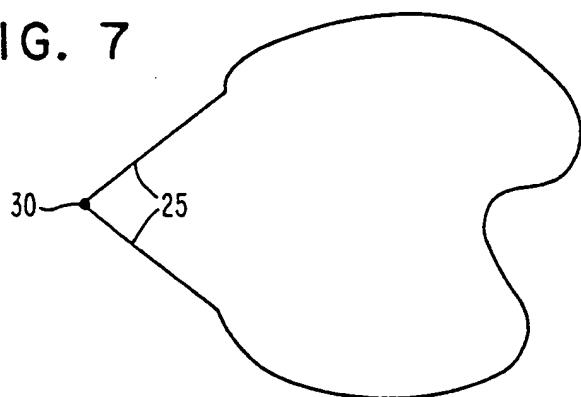


FIG. 8

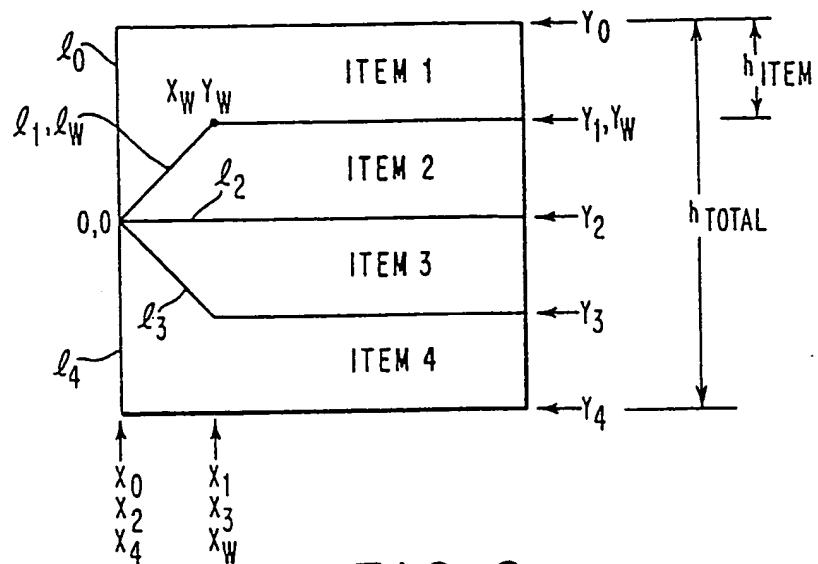


FIG. 9

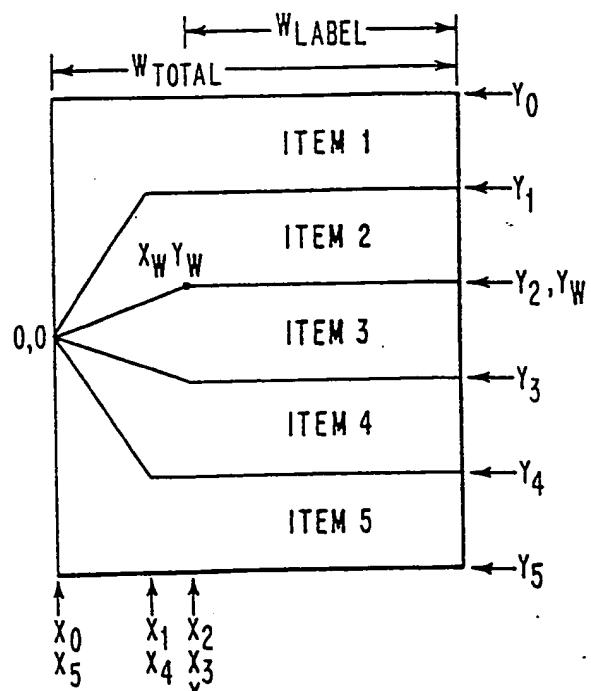


FIG. 10

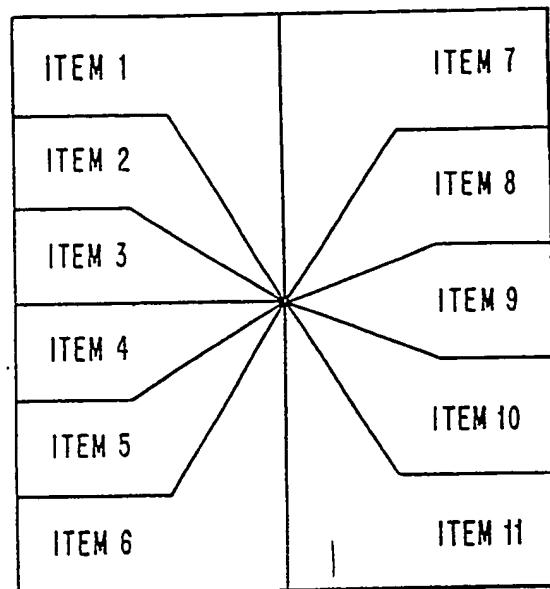


FIG. 11

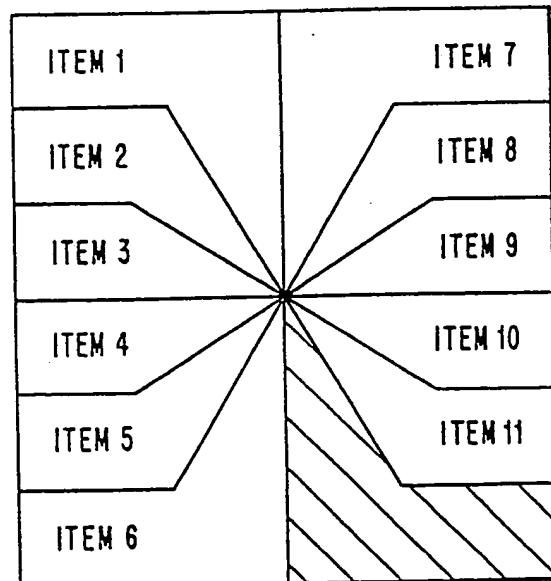


FIG. 12

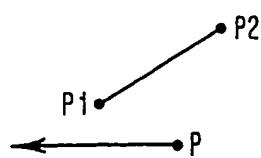


FIG. 13

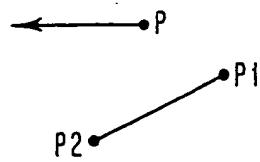


FIG. 14

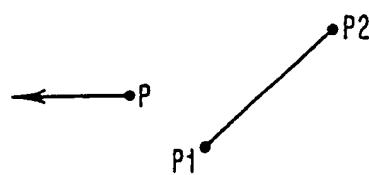


FIG. 15

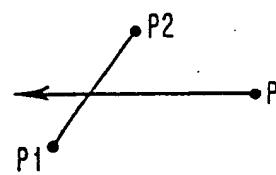


FIG. 16

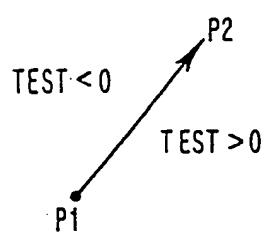


FIG. 17

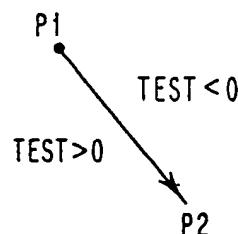


FIG. 18

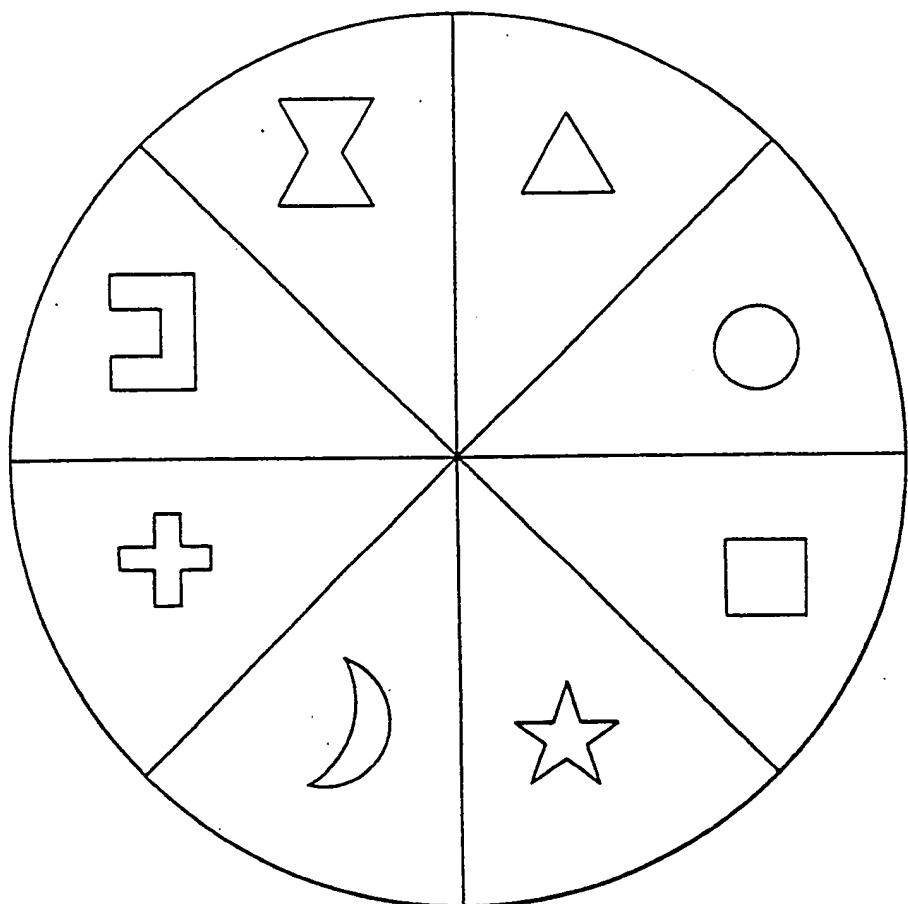


FIG. 19

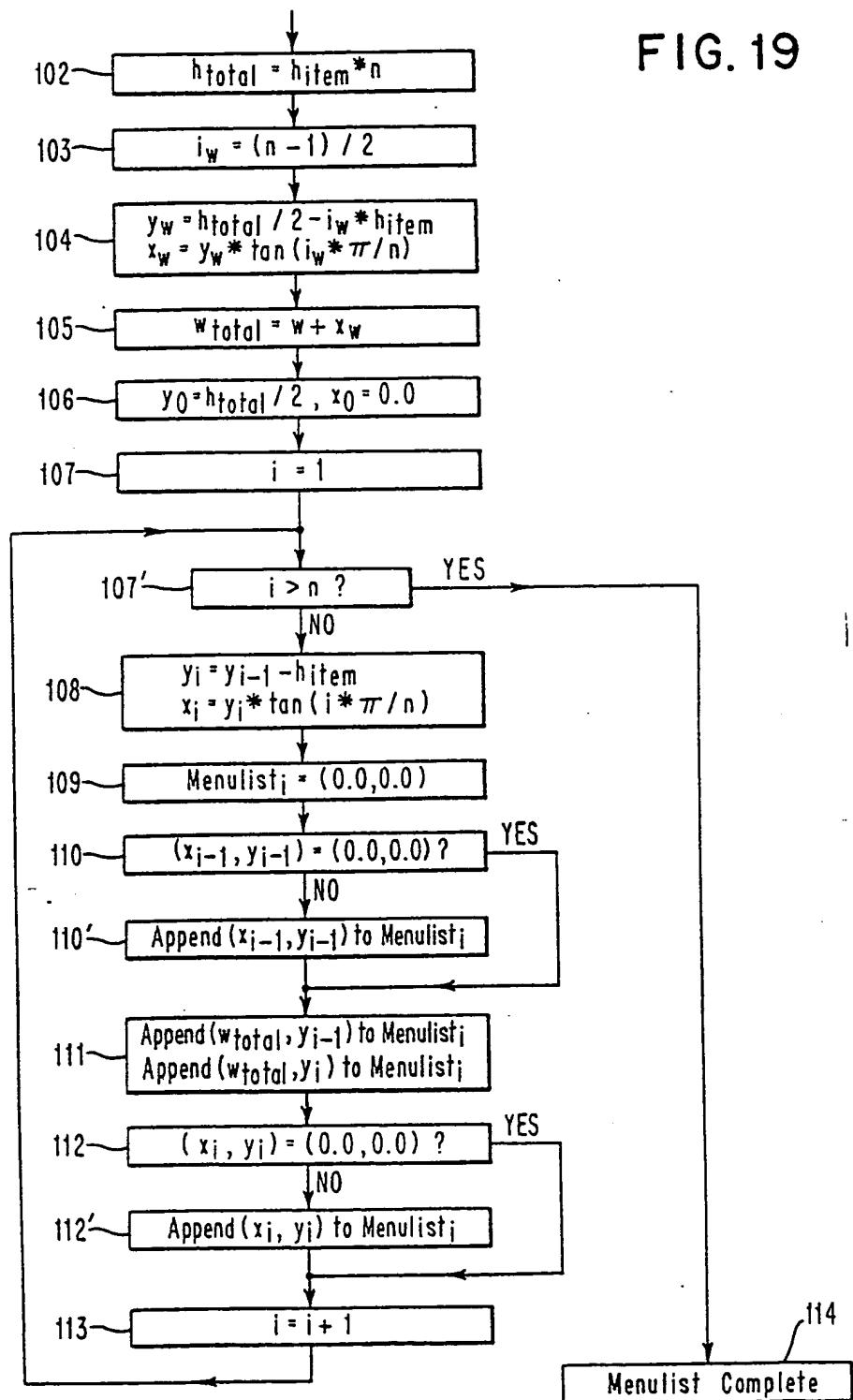


FIG. 20

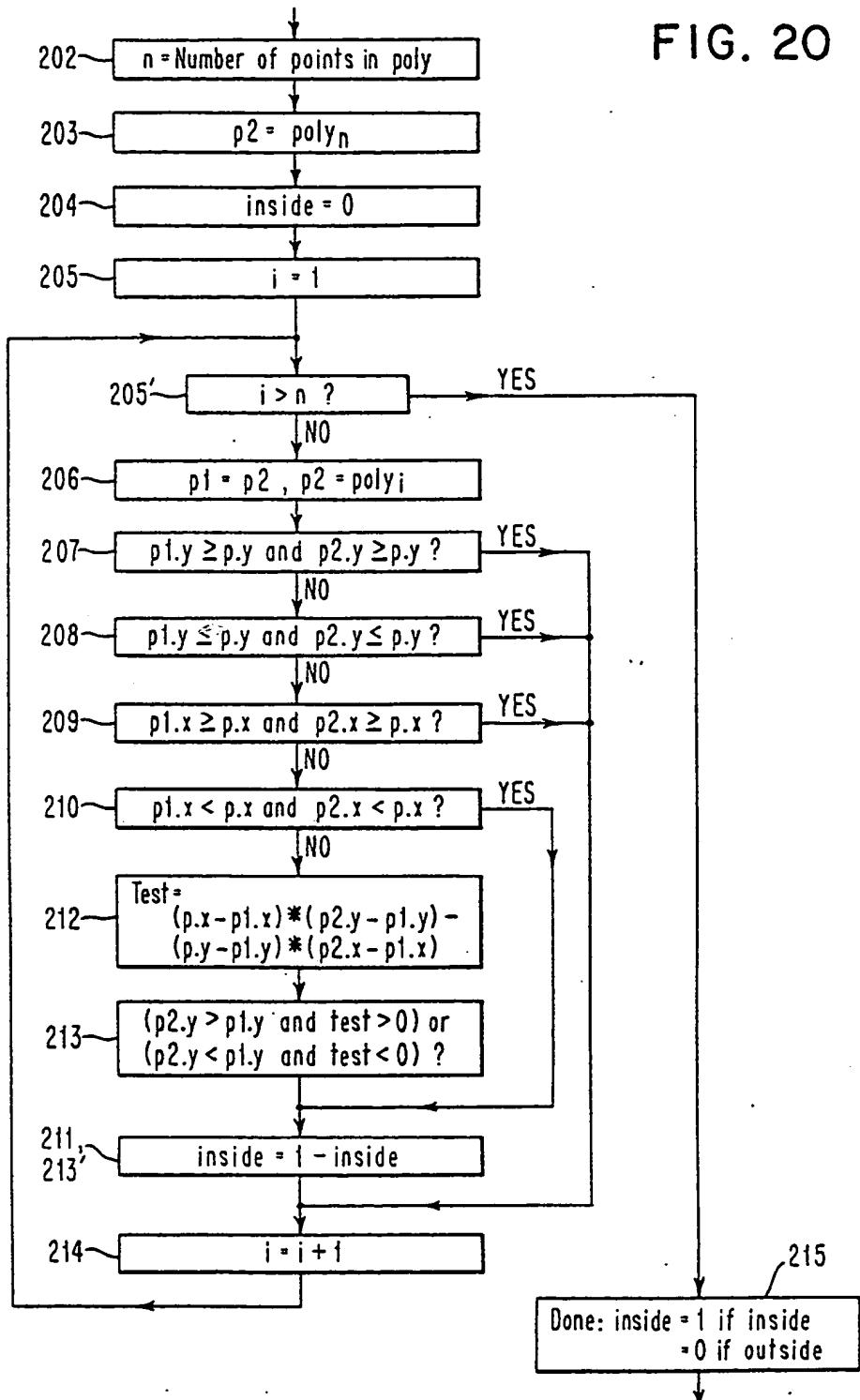


FIG. 21

